П. А. Сеченов, А.А. Оленников, В. П. Цымбал
ФГБОУ ВО «Сибирский государственный индустриальный университет»,
г. Новокузнецк, Россия

ПРИМЕНЕНИЕ ТЕХНОЛОГИИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ CUDA В ЗАДАЧЕ РАСПЛАВЛЕНИЯ ШАРООБРАЗНОЙ ЧАСТИЦЫ

Аннотация

В данной статье рассмотрена технология параллельного программирования CUDA. Показана тенденция современных компьютеров к увеличению мощности за счет увеличения количества ядер. Представлен закон Амдала, позволяющий оценить ускорение времени программы при распараллеливании на N процессоров. Перечислены условия увеличения производительности алгоритма. Представлена задача расплавления частицы железной руды. Рассмотрены особенности языка параллельного программирования CUDA C и представлены алгоритмы для выбранной задачи. Произведен сравнительный анализ времени выполнения задачи на CPU и GPU.

Ключевые слова: параллельное программирование, графический процессор, расплавление частицы, язык CUDA C.

Abstract

This article describes the technology CUDA parallel programming. It is shown that the tendency of modern computers to increase the capacity by increasing the number of cores. Presented Amdama law allows to evaluate the acceleration time when the program parallelization by N processors. Listed the conditions to increase performance of the algorithm. It presents the problem of melting iron ores. The peculiarities of the language of parallel programming CUDA C and presents algorithms for the selected task. A comparative analysis of the execution time of tasks on the CPU and GPU.

Keywords: parallel programming, GPU, melting particles, CUDA C language.

Целью работы является сравнение производительности программы расплавления шарообразной частицы, решаемой на центральном процессоре (CPU) и с применением технологии параллельного программирования CUDA на графическом процессоре (GPU).

Впервые программно-аппаратная архитектура CUDA (Compute Unified Device Architecture) появилась в феврале 2007 года, представив программистам использовать технологию GPGPU (General-Purprose computing on Graphics Processing Units – неспециализированные вычисления на графической карте), благодаря которой на привычных языках высокого уровня (прежде всего – Си) можно реализовывать алгоритмы, которые выполняются на графических ускорителях GeForce восьмого поколения и старше.

Одной из важнейших характеристик любого вычислительного устройство является производительность. Процессоры архитектуры x86, с момента появления в 1978 году, увеличили свою тактовую частоту с 4,77 МГц до 3 ГГц, т.е. более чем в 600 раз, однако в последние несколько лет рост частоты более не наблюдается [1]. Это связанно как с ограничениями технологии производства микросхем, так и с тем, что энергопотребление (а значит и выделение тепла) пропорционально четвертой степени частоты. Таким образом, увеличение тактовой частоты всего в 2 раза приводит к увеличению тепловыделения в 16 раз [2]. В настоящее время рост производительности идёт в основном за счет увеличения числа параллельно работающих ядер, т.е. за счет параллелизма.

Максимальное ускорение, которое можно получить от распараллеливания программы на N процессоров (ядер), дает закон Амдала (Amdahl Law) [2]:

$$S = \frac{1}{(1+P)+\frac{P}{N}} \quad (1)$$

где P – это часть времени выполнения программы, которая может быть распараллелена на N процессоров.

За последние 10 лет появилось множество обучающей литературы по технологии CUDA отечественного [1–3] и зарубежного производства [4; 5]. Также есть обучающий онлайн курс [6] на сайте производителя графических процессоров Nvidia. Автор данного курса производит измерение скорости программ на CPU и GPU как с использованием низкоуровневого программирования на языке CUDA C, так и с использованием программного стандарта параллельного программирования OPENACC, который позволяет программисту абстрагироваться от особенностей инициализации графического процессора, вопросов передачи данных на сопроцессор и обратно. При этом наибольшую производительность для параллельных алгоритмов дает программирование на низкоуровневом языке CUDA C.

Таким образом, для увеличения производительности алгоритма требуется выполнение следующих условий:

- наличие графического процессора с технологией CUDA;
- алгоритм программы должен распараллеливаться;
- наличие среды разработки, в данном случае это Visual Studio 2013;
- наличие компилятора языка CUDA C – nvcc, который входит в состав CUDA SDK;
- знание языков C, CUDA C.

Задача расплавления частицы железной руды [7] хорошо распараллеливается и состоит в расчете скорости расплавления одного слоя:

$$\tau = \frac{\Delta V_{сл} \rho c_p T_{пл}}{\alpha (T_{среды} - T_{слоя}) S}, \quad (2)$$

где $\Delta V_{сл}$ – объем плавящегося слоя; ρ – плотность; c_p – теплоёмкость; $T_{пл}$ – температура плавления; α – коэффициент теплопроводности; $T_{среды}$ – температура среды; $T_{слоя}$ – температура слоя; S – площадь поверхности данного слоя шара.

Перейдем к рассмотрению особенности программной реализации на языке CUDA C. При программировании на GPU используются два понятия: хост, который связан с центральным процессором (host – CPU) и девайс, который связан с графическим процессором (device – GPU). Вызов параллельных частей происходит в функции ядра:

Kernel <<<nBlk, nTid>>> (args),

где: Kernel – название функции ядра;
nBlk – количество блоков функции;
nTid – количество нитей функции;
args – аргументы функции.

В ходе программы на GPU каждая нить имеет свой идентификационный номер:

thredID = threadIdx.x + blockIdx.x * blcockDim.x

где threadIdx – номер нити в блоке;
blockIdx – номер блока, в котором находится нить;
blockDim – размер блока.

Программа состоит из двух функции: основной – Main() выполняем на CPU и функции расчета MyFunc(), выполняемой на GPU.

На рисунке 1 показан алгоритм реализации функции на GPU, который состоит из следующих блоков:

1) вызов функции на GPU с параметрами (массив расчетов, количество расчетов, начальный радиус, шаг, плотность, температура плавления, коэффициент теплоотдачи, температура среды);

2) определение идентификатора нити;

3) определение начального и конечного радиуса;

4) определение времени плавления для каждого слоя, которое рассчитывается параллельно.

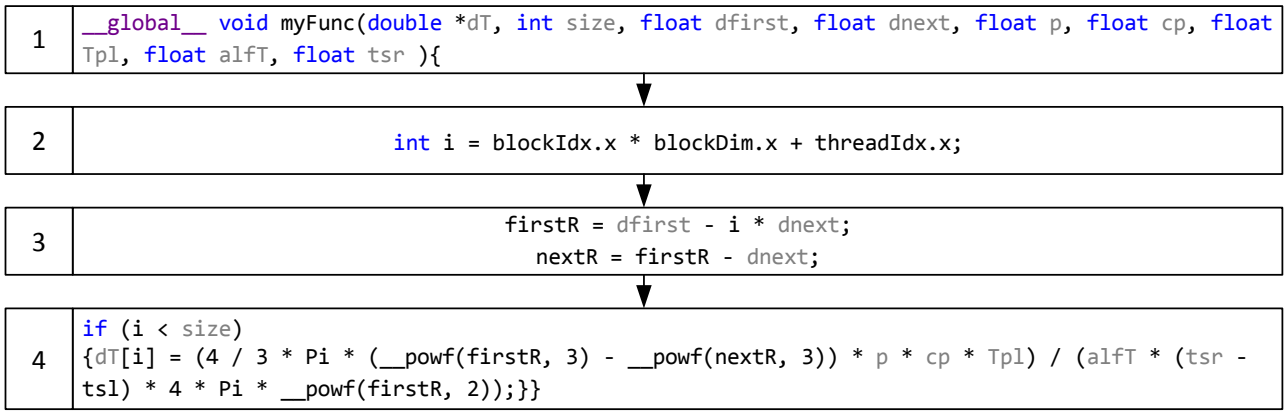

Рис. 1. Алгоритм реализации функции на GPU

На рисунке 2 показан алгоритм реализации функции на CPU, который состоит из следующих стадий:

1) вызов главной функции;
2) выделение памяти для хоста и девайса;
3) определение количества блоков в потоке;
4) копирование данных с хоста (CPU) на девайс (GPU);
5) вызов функции ядра myFunc с требуемыми параметрами;
6) копирование данных с девайса (GPU) на хост (CPU);
7) суммирование времен расплавления слоев;
8) освобождение памяти хоста и девайса.

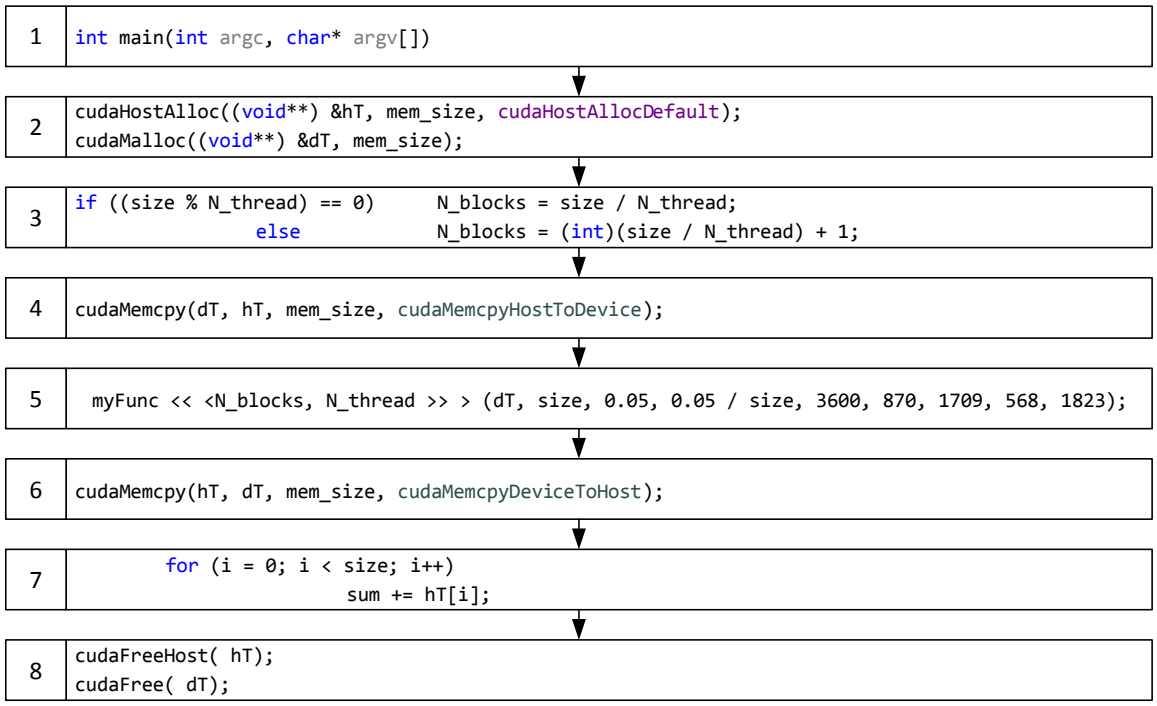

Рис. 2. Алгоритм реализации функции на CPU

В таблице 1 представлена зависимость времени выполнения задачи на CPU и GPU в зависимости от количества слоев, выделяемых в частице.

Таблица 1

Сравнение времени выполнения задачи на CPU и GPU.

Количество слоев	Время выполнения на CPU (1 ядро 3.6 ГГц), с	Время выполнения на GPU (GeForce 560 Ti, 384 ядра), с	Выигрыш в производительности CPU/GPU
10.000	0.00528	0.000462	11.4
100.000	0.0545	0.00129	42.2
1.000.000	0.557	0.00962	57.9
10.000.000	5.778	0.0972	59.4
100.000.000	-	0.782	

Как видно из таблицы 1 выигрыш в производительности GPU составляет от 11.4 до 59.4 раз. Наибольший выигрыш по производительности виден при увеличении количества слоев, так на GPU за одну секунду можно обработать на два порядка больше слоев, чем за тоже время на CPU.

Выводы: технология параллельного программирования CUDA позволяет увеличить производительность распараллеливаемых алгоритмов сложности N до 60 раз. Для этого требуется наличие графического процессора с поддержкой данной технологии, среда разработки и компилятор языка CUDA, знание языка CUDA C, а также хорошее знание задачи и возможности ее распараллеливания. Директива OPENACC не требует знания языка CUDA, но дает меньший прирост производительности в 5–10 раз [6].

Список использованных источников

1. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. – М.: ДМК Прес, 2010. – 232 с.

2. Боресков А.В., Харламов А.А., Марковский Н.Д. и др. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учеб. пособие. – М.: Издательство Московского университета, 2012. – 336 с.

3. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. Слинкина А.А., научный редактор Боресков А.В. – М.: ДМК Пресс, 2011. – 232 с.

4. Cook S. CUDA programming. A Developer’s Guide to Parallel Computing with GPUs. – Morgan Kaufmann, 2013. – 576 p.

5. Cheng J., Grossman M., McKercher T. Professional CUDA C programming. – Wiley, 2014. – 497 p.

6. Перепёлкин Е. Nvidia CUDA и OPENACC. Бесплатный онлайн курс [Электронный ресурс]. Режим доступа: http://www.ispras.ru/conf/2014/nvidia/Lecture_1_v_23_08_2014.webm (Дата обращения 19.04.2016).

7. Сеченов П.А., Оленников А.А. Цымбал В.П. Исследование динамики изменения состава шлака в зонной модели колонного струйно-эмульсионного реактора // В сборнике: Творческое наследие В. Е. Грум-Гржимайло: история, современное состояние, будущее. – 2014. – С. 105-110.